VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Petr Hlavica

Bakalářská práce

Vedoucí práce: Ing. Petr Lukáš, Ph.D.

Ostrava, 2021

Abstrakt

Tato bakalářská práce pojednává o mém působení ve firmě ComProMiS s.r.o., kde jsem absolvoval individuální bakalářskou praxi v posledním ročníku bakalářského studia. V první části této práce je uveden podrobný popis firmy a má pracovní pozice. Dále se práce zabývá vývojem elektronického docházkového systému, především mobilní aplikace pro systém Android, který byl vypracováván v rámci praxe ve vývojovém týmu tvořeném mnou a dalšími studenty Vysoké školy báňské. Na závěr je zhodnocen průběh a přínos praxe.

Klíčová slova

docházkový systém; Android; mobilní aplikace; Xamarin

Abstract

This bachelor thesis discusses my work in company ComProMiS s.r.o., where I participated in an individual professional internship in the last year of bachelor studies. The first part of this work provides a detailed description of the company and my job position. Next parts of this thesis are focused on the development of an electronic attendance system, especially a mobile application for the Android system, which was developed as part of an internship in a development team formed by me and other students of the VSB - Technical University of Ostrava. At the end, the course and benefits of practice are evaluated.

Keywords

attendance system; Android; mobile application; Xamarin

Poděkování

Děkuji Ing. Romanovi Potocznému za umožnění absolvování individuální odborné praxe ve společnosti ComProMiS s.r.o. Dále bych chtěl poděkovat kolegům z firmy Michalovi Hrdému, Janu Vožickému a Danieli Hrtúsovi za odborné vedení a veškerou pomoc.

Mé poděkování patří také Ing. Petru Lukášovi, Ph.D. za rady a věcné připomínky při zpracovávání této bakalářské práce.

Obsah

Seznam použitých zkratek a symbolů

VŠB	–	Vysoká škola báňská
MVC	–	Model View Controler
B2B	–	Business to business
B2C	–	Business to consumer
API	–	Application Programming Interface
CRUD	–	Create, Read, Update, Delete
SaaS	–	Software as a Service
IaaS	–	Intrastructure as a Service
PaaS	–	Platform as a Service
SQL	–	Structured Query Language
ORM	–	Object Relational Mapping
XML	–	eXtensible Markup Language
HTTP	–	Hypertext Transfer Protocol
URI	–	Uniform Resource Identifier
JSON	–	JavaScript Object Notation

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Tato bakalářská práce popisuje mé působení ve firmě ComProMiS s.r.o. Forma vykonání bakalářské práce absolvováním odborné praxe ve firmě pro mě byla atraktivnější než vypracování klasické bakalářské práce zadané školou. Pro tuto formu jsem se rozhodl z několika důvodů, jeden z nich bylo mé přání získat nové praktické zkušenosti a využít znalosti nabyté studiem na vysoké škole v reálném prostředí firmy. Dalšími z důvodů jsou získání povědomí, jak to v takové firmě funguje, možnost vyzkoušet si práci v kolektivu a procvičit si komunikaci v týmu. Také se mi líbila pracovní pozice nabízená touto firmou.

ComProMiS s.r.o. je menší společnost založená roku 1996, která zaměstnává ve svých řadách okolo dvou desítek zaměstnanců a v současné době (2021) sídlí v Ostravě-Kunčičkách. Specializuje se v první řadě na vývoj a správu informačních systémů a tvoří významné projekty pro mezinárodní firmu OTIS a.s. Podrobnější informace o firmě a mé pracovní zasazení jsou uvedeny v následující kapitole 2.

V kapitole 3 je uveden cíl projektu a osnova práce na projektu, jímž bylo vytvořit elektronický docházkový systém v rámci vývojového týmu formovaném studenty VŠB. Další kapitola 4 se zabývá popisem stávajícího chodu evidence docházky ve společnosti. V neposlední řadě jsou v kapitole 5 sepsány mnou použité technologie a postupy, které jsem použil při vývoji systému. U každé technologie je uveden úvod k technologii a případně důvod použití v projektu. V kapitole 6 je následně uveden a popsán návrh databáze. Hlavní částí této práce je návrh, vývoj a testovací nasazení mobilní aplikace pro systém Android (viz kapitoly 7, 8, 9), která je součástí nového docházkového systému. Postup vývoje aplikace je sepsán chronologicky za sebou. V závěrečné kapitole 10 jsou zhodnoceny teoretické a praktické znalosti nabyté studiem při výkonu praxe, nové nabyté znalosti a celkový přínos praxe.

Kapitola 2

ComProMiS s.r.o.

Společnost ComProMiS s.r.o. založená roku 1996 má dlouhodobé zkušenosti s tvorbou informačních systémů, klient-server aplikací, intranetových aplikací, webových stránek a aplikací pro mobilní telefony. V dnešní době se firma soustřeďuje převážně na vývoj v prostředí operačního systému Windows a moderních technologií na bázi .NET Framework v databázových prostředích Microsoft SQL Server nebo Oracle. Většina aplikací firmy cílí do prostředí B2B a B2C, tedy do prostředí podpory nákupu, zpracování zakázek mezi centrálami a pobočkami, včetně podpory nákupu přes moderní portály. Nemalou část aktivit společnosti tvoří i tvorba aplikací pro podporu servisních aktivit. Je také partnerem společnosti Software AG a jako jedni z mála v České republice a na Slovensku umí tvořit software na operačním systému Unix a na bázi databáze Adabas a vývojového prostředí Natural. [1]

2.1 Pracovní zařazení studenta

Ve firmě ComProMiS s.r.o. jsem pracoval na pozici vývojář mobilní aplikace pro mobilní systém Android ve vývojovém týmu tvořeném dalšími třemi kolegy z VŠB, kteří zde také vykonávali odbornou praxi. Úkolem celého našeho vývojového týmu bylo pochopit vedení docházky ve firmě a navrhnout nový elektronický docházkový systém. Mou hlavní úlohou dále bylo graficky navrhnout mobilní aplikaci pro mobilní systém Android a také navrhnout funkce aplikace dle požadavků vedoucího firmy. Poté mobilní aplikaci dle těchto návrhů implementovat ve vývojovém prostředí Visual Studio pomocí technologie Xamarin a programovacího jazyka C#.

Kapitola 3

Cíl projektu

Účelem působení na praxi bylo v rámci vývojového týmu vytvořit nový elektronický docházkový systém, který by sloužil jako základ pro možné budoucí rozšíření pro další společnosti. K tomuto projektu jsme dostali návrh a prototypové řešení webové aplikace z minulého roku, ale z určitých důvodů jsme se rozhodli jej nevyužít a navrhli jsme systém zcela nově.

3.1 Požadavky na nový systém

Jelikož je stávající evidence docházky ve firmě dosti neefektivní a neuhlídatelná (viz kapitola 4), bylo požadavkem vedoucího firmy vytvořit systém, který usnadní fungování firmy v kontextu zaznamenávání příchodů, odchodů a přerušení práce, plánování pracovních plánů, nebo i plánování dovolených, či jiných absencí samotných zaměstnanců. Informační systém by měl také usnadnit práci účetnímu oddělení s počítáním stravenek a odpracovaných hodin pro výpočet mezd. Do systému budou mít přístup pouze zaměstnanci s vytvořeným uživatelským účtem. Požadavkem bylo tedy vytvořit projekt skládající se z následujících podprojektů: 1) webové aplikace, 2) webového rozhraní API, 3) mobilních aplikací pro nejrozšířenější mobilní operační systémy Android a iOS. Jedním z požadavků pro tento projekt bylo využít cloudových služeb nabízených platformou Microsoft Azure.

3.1.1 Webová aplikace

Webová aplikace má sloužit hlavně k administračním operacím, jako jsou schvalování aktivit (viz kapitola 6), lokací pro případný home office a plánovaných absencí zaměstnanců. Dále pak přidávání pracovních plánů pro celou firmu, pobočku, nebo jen zaměstnance a případné generování reportů. Dále musí být schopna zobrazit přehled příchodů, odchodů a plánovaní za dané období. K různým částem aplikace bude povolen přístup pouze na základě přiřazené role uživatele.

3.1.2 Mobilní aplikace

Mobilní aplikace bude sloužit především k zaznamenávání začátku, přerušení a konce práce. Jedna ze žádaných funkcionalit aplikace je kontrola geolokačních údajů, zda se uživatel nachází na určeném místě, podle toho, zda má být přítomen osobně, nebo má zaplánovanou práci z domova. Zaměstnanec by měl v aplikaci mít možnost plánovat své budoucí absence jako jsou dovolené, nebo návštěvy lékaře, ale i pracovní aktivity, jako služební cesty. Měla by také poskytovat uživateli jeho přehled o odpracovaných aktivitách, přesčasech, dovolených a stravenkách. Tyto aplikace budou dvě, jedna pro systém Android a druhá pro systém iOS.

3.1.3 Webové rozhraní API

Webové rozhraní API bude výhradně obsluhovat komunikaci mezi databázovým serverem a mobilními aplikacemi. Na straně API by měly probíhat CRUD (Create, Read, Update, Delete) operace a výpočty pro další využití v mobilní aplikaci.

3.2 Osnova práce ve firmě

3.2.1 Analýza současného vedení docházky a návrh nového systému

Pro vývoj projektu se musel náš tým seznámit se současným vedením docházky ve firmě (viz kapitola 4). Následně jsme po analýze sledovaných aspektů docházky a několika konzultacích vytvořili základní model databáze. Více informací o modelu databáze je uvedeno v kapitole 6.

3.2.2 Seznámení se s cloudovou platformou Azure společnosti Microsoft

Jelikož bylo jedním z požadavků využít cloudové služby platformy Azure, musel jsem si nastudovat práci s touto platformou a najít vhodné služby pro vytvoření databázového serveru a databáze. Dalším úkolem bylo nastudovat si práci se službou Azure DevOps, především s nástrojem Azure Repos, s pomocí kterého byl projekt pravidelně verzován.

3.2.3 Analýza a návrh obrazovek a funkcí mobilní aplikace

Před vývojem obou mobilních aplikací bylo nutné zanalyzovat funkcionalitu aplikace. Podle požadavků jsem následně vytvořil vizi aplikace pro systém Android (viz kapitola 7) a v rámci konzultace tento návrh odprezentoval.

3.2.4 Seznámení se s Entity Framework

Dalším krokem před samotným vývojem bylo vytvoření databáze. Pro tento úkol jsem byl seznámen s Entity Framework a přístupem Code First. V rámci tohoto úkolu jsem vytvořil třídy reprezentující

tabulky databáze a následně za pomocí přístupu Code First vytvořil tabulky a vztahy v databázi na platformě Azure (viz podkapitola 8.2).

3.2.5 Vývoj mobilní aplikace pomocí technologie Xamarin

V rámci vývoje jsem se musel seznámit s technologií Xamarin. Vývoj probíhal po většinu času stráveném na praxi. Při vývoji mi byla každý týden určena část, kterou bych měl naprogramovat. Postup při programování byl pravidelně konzultován. Důležitou součástí vývoje byla komunikace s kolegou, který vytvářel API. Vývoj je podrobněji popsán v podkapitole 8.3.

3.2.6 Testování a ladění aplikace

Posledním úkolem bylo nasazení aplikace do testovacího provozu a oprava chyb (viz kapitola 9), které se při testování vyskytly. Chyby byly následně opravovány podle zpětné vazby testovací skupiny.

Osnova	**Časová náročnost**
Analýza současného vedení docházky a návrh nového systému	6 dní
Seznámení se s cloudovou platformou Azure	2 dny
Analýza a návrh obrazovek a funkcí aplikace	5 dní
Seznámení se s Entity Framework	1 den
Vývoj mobilní aplikace	26 dní
Testování a ladění aplikace	10 dní

Tabulka 3.1: Časová náročnost jednotlivých úkolů

Kapitola 4

Popis dosavadní evidence docházky

V této kapitole budu popisovat vedení docházky ve firmě ComProMiS s.r.o., jenž je důležitou součástí pro výpočet platů zaměstnanců. V dnešní době se docházka společnosti zanáší do knihy příchodů a odchodů. Za zápis do této knihy jsou odpovědni sami zaměstnanci. Na začátku každého měsíce účetní provádí součet zaevidovaných hodin každého pracovníka a poté přepis vypočtených hodnot do tabulkového procesoru, z nichž se následně vypočítají jednotlivé mzdy a také počty stravenek. Pro plánování dovolených a pracovních cest se využívá webového kalendáře.

4.1 Fond pracovní doby

Společnost ComProMiS s.r.o. má fond pracovní doby[1] stanoven na 37,5h týdně s pružnou pracovní dobou. Od 9:00 hodin do 14:00 hodin je základní pracovní doba, kdy je zaměstnanec povinen být na pracovišti. V dobách od 6:00 hodin do 9:00 hodin a od 14:00 hodin do 18:00 hodin je pracovní doba volitelná. Zaměstnanec si tedy volí, jak bude ve volitelné pracovní době pracovat podle svého uvážení, ale na konci každého týdne by měl součet odpracovaných hodin splňovat týdenní pracovní fond. Časy docházky se zaokrouhlují po 15-ti minutách. V případě příchodů se čas zaokrouhluje nahoru, to znamená, že pokud přijde zaměstnanec v 7:05, tak je jeho čas příchodu po zaokrouhlení 7:15. Při časech odchodů se zaokrouhluje dolů, tedy v případě že zaměstnanec odejde z práce v 14:10, jeho pracovní doba končí ve 14:00. Pokud nesplní základní pracovní dobu, například kvůli pozdnímu příchodu, nebo brzkému odchodu, je povinen využít půldenního volna. Jakmile pracovník splní pracovní dobu nejméně 5 hodin za den, má nárok na půlhodinovou přestávku a podle vnitřního předpisu má také nárok na stravenku.

[1] Fond pracovní doby je doba, kterou musí zaměstnanec strávit v práci.

4.2 Absence

4.2.1 Dovolená

Pracovníci firmy mají právo na 25 dní dovolené v rámci roku. Dny dovolené by měl zaměstnanec v daném roce využít, pokud se tak však nestane, může si určitý počet dní převést do dalšího roku. Z počtu dní dovolené se odečítají půldenní a celodenní volna. Totožně jako u pracovní neschopnosti se fond povinné pracovní doby sníží o počet dnů čerpaného pracovního volna.

4.2.2 Náhradní volno

Zaměstnanec, který pracuje přesčas v pracovní den, nebo pracuje o víkendu, na dovolené, či o svátcích, si může vzít náhradní volno.

4.2.3 Osobní překážky zaměstnance

Mimo docházku se také evidují osobní překážky na straně zaměstnance, u kterých, pokud to zákon dovoluje, má zaměstnanec nárok na plnou, nebo částečnou náhradu mzdy. U každé z těchto absencí je nutno doložit odpovídající doklad. Mezi tyto překážky patří zejména:

- **Návštěva lékaře, či darování krve** – Celodenní návštěva lékaře nebo darování krve se počítá jako odpracovaný den. Krátká absence z důvodu návštěvy lékaře se počítá jako odpracovaný čas.
- **Jiné důvody** – Jako jiné důvody se berou například svatba, pohřeb, stěhování a další důvody uvedeny v zákoníku práce[2].
- **Pracovní neschopnost** – Zaměstnanec dokládá potvrzení o pracovní neschopnosti, jestliže byl uznán příslušným lékařem dočasně neschopným k výkonu práce. Fond pracovní doby je snížen o onu dobu, po kterou je pracovník uznán nemocný.

4.2.4 Ostatní okolnosti

- **Home office** – Pracovník není přítomen na pracovišti, ale vykonává práci z domova. Počítá se jako odpracovaný den.
- **Služební cesta** – Považuje se jako odpracovaný den. Práce ve stejný den mimo služební cestu se počítá jako hodiny navíc.
- **Státní svátek** – Počítá se jako odpracovaný den. Jestli zaměstnanec pracuje přes státní svátek, má možnost si vzít náhradní volno.

[2]Zákoník práce je zákoník upravující převážnou část českého individuálního pracovního práva. [2]

Kapitola 5

Použité technologie

5.1 Microsoft Azure

Azure je cloudová platforma vytvořená a vyvíjená společností Microsoft, která je využívána pro hostování, vývoj a škálování aplikací skrze datová centra Microsoftu. Platforma nabízí široké spektrum služeb a podporuje různé programovací jazyky, nástroje a rámce. [3] Hlavní typy služeb nabízených touto platformou:

- **Software jako služba (SaaS)** – Model poskytování software, který umožňuje uživatelům připojit se k aplikaci prostřednictvím internetu. Veškerá podpůrná infrastruktura, middleware [1], software a data aplikace je umístěna v datovém centru poskytovatele služeb. [5]

- **Platforma jako služba (PaaS)** – Úplné prostředí pro vývoj a nasazení v cloudu, zahrnující servery, úložiště, sítě, operační systémy, ale také nástroje pro vývoj, správu databází a testování. [6]

- **Infrastruktura jako služba (IaaS)** – Předem připravená výpočetní infrastruktura, která je poskytovaná a spravovaná přes internet. [7]

Prvním využitím platformy Azure byla služba **Azure DevOps**, přesněji nástroj této služby **Azure Repos**, který slouží k verzování projektu pomocí privátního úložiště Git. Důvodem využití bylo zjednodušení práce se sdíleným projektem. Jelikož byla k informačnímu systému potřeba databáze, bylo využito i služeb **databázového serveru** a **databáze SQL Server**. Posledním využitím Azure byla služba **Visual Studio App Center**. Tato služba nabízí vývojářské nástroje pro vytváření, testování, vydávání a monitorování mobilních a desktopových aplikací. [8]

[1] Middleware je specializovaný software, který poskytuje aplikacím služby nad rámec služeb poskytovaných operačním systémem. [4]

5.2 Microsoft Visual Studio

Visual Studio je vývojové prostředí společnosti Microsoft. Toto vývojové prostředí podporuje velké množství různých programovacích jazyků a využívá se tak pro tvorbu počítačových programů, webových stránek, webových aplikací, webových služeb a mobilních aplikací. [9] Celé řešení docházkového systému bylo vytvářeno v tomto vývojovém prostředí. Výhodou tohoto prostředí je i poskytované uživatelské rozhraní Git pro jednoduché verzování projektů a podpora již zmíněné služby Azure Repos.

5.3 SQL Server Management Studio

SQL Server Management Studio je softwarová aplikace společnosti Microsoft, která se využívá ke konfiguraci, správě a administraci databáze Microsoft SQL Server. [10] Tato aplikace byla využívána pro přístup do databáze a v počátcích vývoje především ke kontrole a k mazání vložených dat. Dále byla využívána pro zkoušení různých dotazů při návrhu funkcí aplikace.

5.4 Microsoft SQL Server

Microsoft SQL Server je relační databázový systém vyvinutý společností Microsoft. Primární funkcí tohoto systému je ukládání a načítání dat podle požadavků jiných softwarových aplikací. [11] Tento databázový systém je v dnešní době největším konkurenčím produktem databázových systémů MySql a Oracle od společnosti Oracle.

5.5 Entity Framework

Entity Framework je ORM rámec s otevřeným zdrojovým kódem pro .NET aplikace podporovaný společností Microsoft. Umožňuje vývojářům pracovat s daty pomocí objektů doménových tříd bez nutnosti zaměření na databázové tabulky a sloupce, kde jsou tato data uložena. S Entity Framework mohou vývojáři při práci s daty pracovat na vyšší úrovni abstrakce a mohou tak vytvářet a udržovat datově orientované aplikace s menším psaním kódu, jelikož tento rámec dokáže spoustu tříd vygenerovat. [12] Entity Framework podporuje tři přístupy:

- **Code First** – Tento přístup spočívá v tom, že se nejdříve naprogramují třídy představující tabulky databáze, na základě kterých se pomocí Entity Framework vytvoří struktura databáze.
- **Database First** – S pomocí tohoto přístupu lze na základě již vytvořené databáze pomocí Entity Framework vytvořit třídy reprezentující tabulky databáze.
- **Model First** – Tímto přístupem lze z vytvořeného modelu databáze nástrojem Entity Framework Designer vytvořit databázi a třídy reprezentující vymodelované tabulky.

Jelikož se firma v dnešní době zaměřuje na vývoj projektů pomocí přístupu Code First, byl mi tento přístup vysvětlen a pomocí tohoto přístupu jsem následně vytvořil databázi (viz podkapitola 8.2).

5.6 Programovací jazyk C#

C# je vysokoúrovňový objektově orientovaný jazyk vyvinutý firmou Microsoft zároveň s platformou .NET Framework. Microsoft založil C# na jazycích C++ a Java (a je tedy nepřímým potomkem jazyka C, ze kterého čerpá syntaxi). C# lze využít k tvorbě databázových programů, webových aplikací a stránek, webových služeb, formulářových aplikací ve Windows, softwaru pro mobilní zařízení (PDA a mobilní telefony) atd. [13]

5.7 Xamarin

Xamarin je platforma s otevřeným zdrojovým kódem pro vytváření moderních aplikací pro iOS, Android a Windows pomocí .NET. Je to abstraktní vrstva, která spravuje komunikaci sdíleného kódu se základním kódem platformy. [14]

5.8 Proto.IO

Proto.IO je nástroj pro prototypování aplikací vytvořený společností PROTOIO Inc. Proto.IO využívá uživatelské rozhraní drag and drop a nevyžadujé kódování. [15] Tento nástroj byl použit pro grafický návrh obrazovek mobilní aplikace v rámci patnáctidenní zkušební lhůty.

5.9 Creately

Creately je nástroj vizuální spolupráce s možnostmi vytváření diagramů a návrhů vytvořený společností Cinergix. [16] Nástroj byl využíván celým vývojovým týmem pro tvorbu modelu databáze. Důvodem bylo jednoduché sdílení a možnost pracovat na modelu současně ve více lidech.

5.10 Microcharts

Pro grafické zobrazení dat pomocí grafů jsem využil volně dostupnou knihovnu Microcharts [17], kterou jsem do projektu přidal skrze správce balíčků NuGet [18], jenž je součástí nástroje Visual Studio. Knihovna mě zaujala svou jednoduchostí použití a nebylo tak nutné tvořit vlastní grafy. Z této knihovny jsem použil prstencový graf, který je využíván na čtyřech obrazovkách.

5.11 Iconify

Jelikož bylo navrhováno použití ikon jako designového prvku aplikace, využil jsem další z volně dostupných balíčků Iconify [19]. Tento balíček nabízí využití různých sad ikon, které jsou ve formě fontů. Výhodou těchto ikon je, že je programátor nemusí stahovat ve formě obrázků z internetu. Ikony, které jsou zobrazovány v aplikaci využívají tento balíček.

Kapitola 6

Návrh databáze

Po pochopení vedení docházky ve společnosti a analýze požadavků na nový elektronický docházkový systém přišel čas na navržení vhodného databázového modelu. Pro počáteční vývoj jsme vytvořili základní model. V průběhu praxe byla databáze měněna a rozšiřována o další tabulky. V této kapitole bude tedy popisován vysledný model databáze.

6.1 Popis tabulek

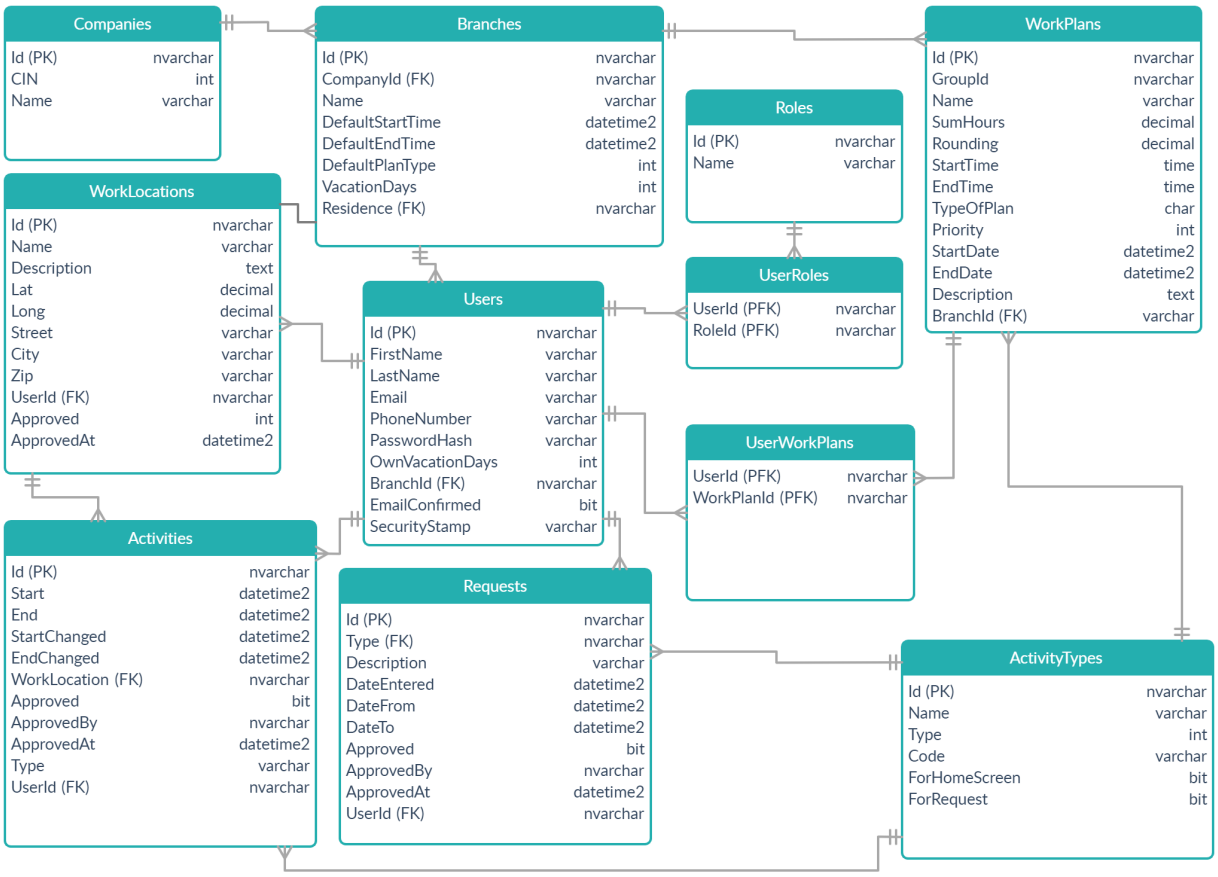
Následující popis se týká tabulek, které můžeme vidět ve finálním modelu databáze na obrázku 6.1. Nejdůležitějšími tabulkami jsou `Activities` a `WorkPlans`. Aktivita uchovávána v tabulce `Activities` je část dne zaměstnance započatá a ukončená v mobilní aplikaci. Uchovává údaje o jejím začátku, konci, lokaci, typu a statusu schválení. Pro další možnosti uživatele webové aplikace s odpovědnou rolí jsou zde atributy, které slouží pro změnu začátku, či konce dané aktivity při schvalovacím procesu v systému. Pracovní plány evidované v tabulce `WorkPlans` slouží jako předpis aktivit zaměstnanců a poboček. Plány pracují na principu prioritizace. Hodnoty priorit jsou následovné:

- **0** – Základní plán pro celou firmu určující fond pracovní doby a povinnou pracovní docházku.
- **1** – Plány pro státní svátky.
- **2** – Vedlejší plán pro celou firmu nebo pro pobočku, který má jiné údaje než plán s hodnotou priority 0.
- **3** – Plány zaměstnanců pro osobní pracovní aktivity, lišící se od plánů s prioritami 0 nebo 2, dovolené a jiné absence.

Jednou z důležitých tabulek je i tabulka `Requests` pro uchovávání žádostí zaměstnanců. Žádost zadává zaměstnanec přes aplikaci v případě plánování dovolených, návštěv lékaře, pracovních cest atd. Na základě těchto žádostí jsou po schválení ve webové aplikaci vytvořeny v databázi plány pro

daného zaměstnance. Pro uchovávání typů aktivit, plánů a žádostí byla v průběhu praxe navržena tabulka `ActivityTypes`. Typy aktivit uchováváné v této tabulce se dělí na produktivní, neproduktivní a absence. Produktivní aktivita je práce v kanceláři, home office a služební cesta. Neproduktivní aktivity jsou osobní překážky zaměstnance, které jsou proplaceny (viz podkapitola 4.2.3). Absence je například dovolená, pauza na oběd nebo jiné přerušení práce. Atributy `ForHomeScreen` a `ForRequest` byly přidány později pro funkce API, které jsou využívány v mobilních aplikacích. Tabulka `Companies` uchovává údaje o firmě. Tato tabulka je důležitá pro možné využití systému více firmami. Pro nastavení systému pro jiné pobočky firmy byla navržena tabulka `Branches`. Slouží pro uchovávání údajů o základní pracovní době pobočky, názvu, lokalitě a předepsaných dnech dovolené pro zaměstnance pobočky. Lokace zaměstnanců a poboček jsou uchovávány v tabulce `WorkLocations`. Součástí jsou i atributy pro schvalování lokací zaměstnanců. Lokace se využívají v aplikaci pro kontrolu, zda je uživatel na sjednaném místě. V tabulce `Users` jsou uchovávány základní údaje o zaměstnancích. Pro zjednodušení práce s databází je u každého zaměstnance uložena hodnota s počtem zbývajících dnů dovolené. Mimo to jsou zde i atributy pro využití ASP.NET Identity[1] ve webové aplikaci a webovém rozhraní API. Tabulky `Roles` a `UserRoles` jsou určeny k přiřazení role uživateli.

[1] ASP.NET Identity je systém členství pro ověřování a autorizaci uživatelů v aplikacích ASP.NET. [20]



Obrázek 6.1: Model databáze

Kapitola 7

Návrh mobilní aplikace

Jedna z důležitých částí vývoje mobilní aplikace byl její grafický návrh. Především bylo nutné zkonzultovat, jaké zobrazované informace jsou relevantní pro uživatele, v jaké části aplikace by se měly zobrazovat a jak by měly obrazovky být na sebe vázány. Po ujasnění těchto náležitostí a inspirování aplikacemi stejného typu, jsme se dohodli, že aplikace bude mít menu pro navigaci mezi čtyřmi hlavními obrazovkami. Původně byly vytvořeny návrhy dva. Jeden pro Android aplikaci a druhý pro iOS aplikaci, který vytvářel kolega. Z důvodu preference vedoucího byly návrhy sjednoceny a jako předloha pro vývoj obou aplikací se využíval návrh vytvořený mnou.

7.1 Obrazovka „Docházka“

Domovská obrazovka pojmenována jako „Docházka“ (viz obrázek 7.1a) zobrazuje důležité denní informace. Hlavní je informace o plánu přihlášeného zaměstnance, respektive co má naplánované na daný den (např. home office). Další informace je o pracovní době, ve které by měl pracovat. Poté by měla být viditelná informace, kdy zaměstnanec začal pracovat a celkový odpracovaný čas od začátku pracovní aktivity. Součástí této obrazovky je i graf pro grafické zobrazení odpracované doby podle fondu pracovní doby. Zbytek obrazovky tvoří tlačítka pro začátek či konec práce, čerpání půldenní a celodenní dovolené a tlačítko pro přerušení práce. V prvních návrzích se počítalo se třemi grafy pro zobrazení odpracované doby za den, týden a měsíc a chyběly informace o začátku pracovní doby. Tyto grafy se nakonec přesunuly na jinou obrazovku. Původně mělo existovat také více tlačítek, ale z hlediska vzhledu a komplikovanosti jsme se dohodli jen pro zmiňovaná čtyři tlačítka.

7.2 Obrazovka „Přehled“

Druhou obrazovkou je „Přehled“ (viz obrázek 7.1b), která zastřešuje tři karty pro denní, týdenní a měsíční přehled práce zaměstnance. První z karet poskytuje uživateli rolovací seznam již ukončených pracovních aktivit a přerušení práce. Seznam je setříděn dle data a času od nejnovějších

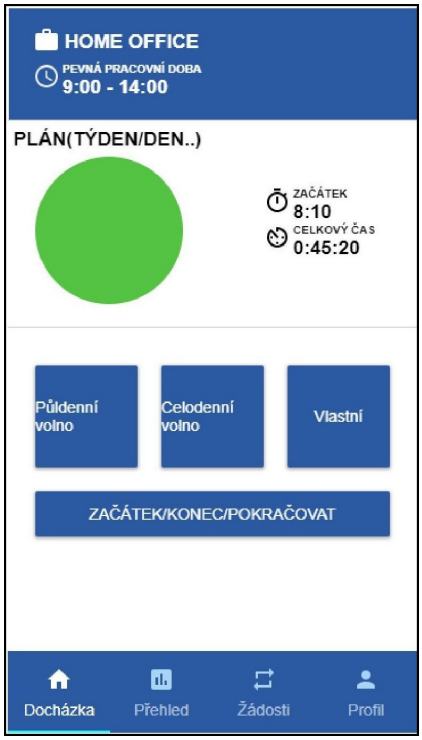
aktivit po nejstarší. U těchto aktivit má uživatel přehled o jakou aktivitu se jednalo, čas začátku a konce, dobu trvání a ikonu, která odpovídá statusu schválení dané aktivity. Po rozkliknutí aktivity se uživateli zobrazí obrazovka detailu aktivity, která skrývá více informací o dané aktivitě a lokaci na které byla vykonávána. Další dvě karty pro týdenní a měsíční přehled jsou skoro totožné. Obě obsahují graf, který zobrazuje poměr vykonané práce, využité dovolené, nebo náhradního volna. Ve finálním řešení aplikace byl do tohoto poměru přidán i počet dní nemocenské, se kterým se v návrhu nepočítalo. Graf zobrazuje hodnoty podle sledovaného období (týdne nebo měsíce). Pod grafem je seznam informací o odpracované době, o přesčasech, či době, která zbývá dopracovat a počet čerpaných dní dovolené za dané období. Měsíční přehled nabízí uživateli navíc informaci o počtu stravenek, na které má nárok za odpracované hodiny v daném měsíci. Nad těmito přehledy je i lišta pro přepínání týdnů nebo měsíců.

7.3 Obrazovka „Žádosti“

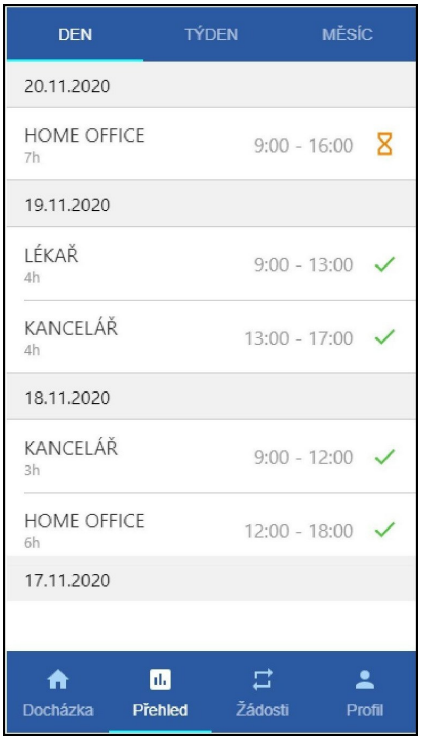
Další obrazovka je v menu pod názvem „Žádosti“ (viz obrázek 7.1c). Tato obrazovka, stejně jako ta minulá, v sobě skrývá systém karet. Zde jsou pouze karty dvě, a to karta „Nadcházející“ a „Historie“. Systém karet byl zde zvolen pro přehlednost. Každá karta obsahuje seznam podaných žádostí daného zaměstnance setřízených podle data, kdy se má požadovaná skutečnost konat. V seznamu se zobrazuje název žádosti, datum nebo období, na kdy je naplánována a ikona, která jako u denního přehledu udává status schválení. Součástí této obrazovky je i tlačítko pro otevření formuláře nové žádosti.

7.4 Obrazovka „Profil“

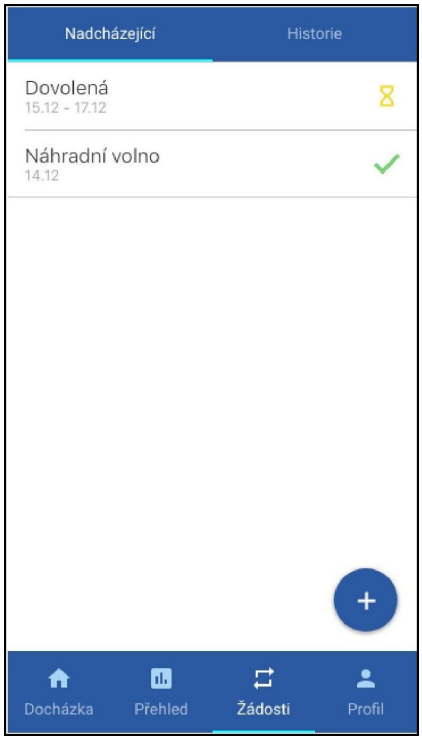
Poslední obrazovkou v menu je „Profil“ (viz obrázek 7.1d). Zde je uvedeno jméno přihlášeného zaměstnance, jeho e-mail a telefonní číslo. Následuje graf pro přehled zaměstnancovy zbývající, čerpané a plánované dovolené. Pod grafem je menu s volbami pro otevření obrazovky s lokacemi zaměstnance, otevření dialogu pro změnu hesla a tlačítko pro odhlášení z aplikace. Na obrazovce s lokacemi má uživatel přehled o svých lokacích přidaných do systému. U každé lokace je jim uvedený název lokace, adresa a ikona statusu schválení lokace. Stejně jako u obrazovky se žádostmi je zde tlačítko pro otevření obrazovky s formulářem na přidání nové lokace.



(a) Návrh obrazovky „Docházka“



(b) Návrh obrazovky „Přehled“



(c) Návrh obrazovky „Žádosti“

(d) Návrh obrazovky „Profil“

Obrázek 7.1: Finální návrhy obrazovek

Kapitola 8

Vývoj

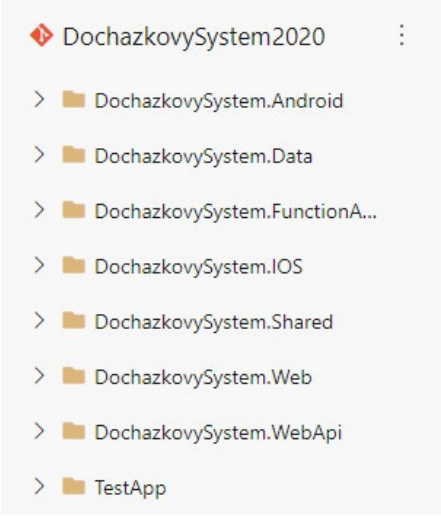
Tato kapitola pojednává o částech nového docházkového systému, které jsem řešil převážně sám. Stěžejní část této kapitoly se věnuje implementaci mobilní aplikace docházkového systému pro systém Android.

8.1 Struktura projektu

Před vývojem všech částí systému byl důležitým krokem samotný projekt vytvořit a vymyslet jeho strukturu. Po konzultaci s kolegy z firmy jsme strukturu projektu vytvořili podle ukázkového projektu firmy z minulých let. Poté bylo nutné námi vytvořenou strukturu propojit s repositářem ve službě Azure Repos, kde byl celý projekt během praxe verzován. Struktura celého projektu následuje logické členění uvedené v podkapitole 3.1. Tuto strukturu základních složek projektu můžeme vidět na obrázku 8.1. Popis jednotlivých podprojektů této struktury je následovný:

- **`DochazkovySystem.Android`** – Podprojekt mnou vyvíjené aplikace pro mobilní operační systém Android. Více informací o vývoji aplikace je uvedeno v podkapitole 8.3.
- **`DochazkovySystem.Data`** – Knihovna tříd obsahující třídy reprezentující tabulky databáze a vygenerované třídy pomocí Entity Framework (viz podkapitola 8.2). Dále tato knihovna tříd obsahuje třídu `ObjectMapper` pro mapování objektů na jiné objekty a třídy, které obsahují metody zajišťující komunikaci s databází pro další využití v podprojektu webového rozhraní API.
- **`DochazkovySystem.FunctionApp`** – Podprojekt, který obsahuje kód samospustitelné služby pro odpočet zbývajících dní dovolené.
- **`DochazkovySystem.IOS`** – Podprojekt aplikace pro mobilní operační systém iOS.
- **`DochazkovySystem.Shared`** – Knihovna tříd obsahující viewmodely, výčtové typy `Enum` a třídy sloužící pro propojení mobilních aplikací s rozhraním API (viz podkapitola 8.3.3).

- **`DochazkovySystem.Web`** – Podprojekt ve kterém je implementována webová aplikace docházkového systému.
- **`DochazkovySystem.WebApi`** – Podprojekt webového rozhraní API, které poskytuje služby mobilním aplikacím.
- **`TestApp`** – Podprojekt konzolové aplikace, která sloužila v počátcích vývoje pro testování služeb webového rozhraní API.



Obrázek 8.1: Struktura projektu v Azure Repos

8.2 Vytvoření databáze pomocí Entity Framework

Prvním mým úkolem bylo vytvoření tříd reprezentujících tabulky databáze v knihovně tříd `DochazkovySystem.Data` (viz obrázek 8.1). Na konzultaci s kolegou z firmy mi následně byl vysvětlen přístup Code First (viz podkapitola 5.5) pro vytvoření databáze. Tento přístup dovoluje vývojářům tvořit databázi pomocí migrací, což je popis změn mezi verzemi tříd reprezentující tabulky databáze, který má dopad na schéma databáze. [21]. Pro tento přístup jsem musel danou knihovnu tříd propojit s databází a vytvořit třídu `DataContext` pro nastavení migrací. Následně s pomocí příkazů v konzoli Visual Studio byla vytvořena databáze. Tyto příkazy jsou tři:

- **`Enable-Migrations`** – Příkaz vygeneruje třídu `Configuration` a povolí se tak migrace ve zvoleném projektu.
- **`Add-Migration`** – Po zadání tohoto příkazu s názvem migrace se vygeneruje třída s časovou značkou[1] a názvem migrace, tato třída obsahuje metody `Up` a `Down`. Metoda `Up` obsahuje

[1] Časová značka v případě migrací je sekvence znaků vyjadřující datum a čas vzniku dané migrace.

operace, které se vykonají v případě migrace na vyšší verzi. Naopak metoda `Down` obsahuje operace, které se vykonají v případě vrácení se zpět do předchozího stavu. Pokud tedy metoda `Up` obsahuje operaci pro vytvoření například tabulky databáze, tak metoda `Down` obsahuje operaci, která tuto tabulku smaže v případě vrácení se na nižší verzi.

- `Update-Database` – Tímto příkazem se provedou změny v databázi vykonáním metod vytvořených ve třídě vygenerované předchozím příkazem.

8.3 Vývoj mobilní aplikace pro systém Android

V této kapitole je popsán vývoj mobilní aplikace pro systém Android, která je součástí nového elektronického docházkového systému. Níže uvedené podkapitoly jsou seřazeny chronologicky podle toho, jak jsem postupoval při vývoji této aplikace.

8.3.1 Tvorba uživatelského rozhraní

Na začátek vývoje samotné aplikace jsem dostal za úkol vytvořit uživatelské rozhraní aplikace podle návrhu uvedeného v kapitole 7. Jelikož jsem se s platformou Xamarin setkal poprvé, musel jsem si nastudovat některé časti z její dokumentace. Na vizuální styl aplikace nebyly žádné speciální nároky, a tak jsem jej mohl udělat podle sebe. Při vytváření rozložení jsem využíval návrhového systému Material Design [22] a snažil jsem se vzhled vytvářet pomocí komponent tohoto systému a podle jeho pravidel[2]. Pro vizualizaci jsem v počátcích vývoje využíval statická data.

Android aplikace jsou tvořeny třídami aktivit, které poskytují hlavní zobrazení aplikace. Po vytvoření projektu jsem měl k dispozici jednu třídu hlavní aktivity aplikace `MainActivity`. Ostatní obrazovky aplikace jsem tvořil jako fragmenty, které představují opakovaně použitelnou část uživatelského rozhraní [23], neboli třídy dědící ze třídy `Fragment`. XML soubor, který reprezentuje rozložení hlavní aktivity obsahuje pouze menu aplikace a komponentu `FrameLayout` pro zobrazování čtyř hlavních fragmentů, neboli obrazovek z návrhu. Ve třídě hlavní aktivity je potom naprogramována inicializace a zobrazování, či schovávání těchto fragmentů při kliknutí na položky menu pomocí správce fragmentů `SupportFragmentManager`.

Pro obrazovku přihlášení jsem vytvořil třídu aktivity `LoginActivity`. Tato obrazovka obsahuje dvě textová pole pro zadání přihlašovacích údajů a tlačítko pro příhlašení. Ve třídě této obrazovky probíhá i kontrola vstupů.

Prvním fragmentem v menu je fragment obrazovky „Docházka“ (viz kapitola 7.1 a obrázek 7.1a). Tento fragment slouží jako hlavní obrazovka aplikace. V horní části fragmentu jsou nad sebou komponenty `ImageView` pro ikony a vedle nich komponenty `TextView` pro informace o pracovním plánu a pevné pracovní době. Pod těmito informacemi je komponenta rozložení `CardView`, která zastřešuje prstencový graf z balíčku Microcharts (viz kapitola 5.10) pro zobrazení odpracovaných hodin. Dále

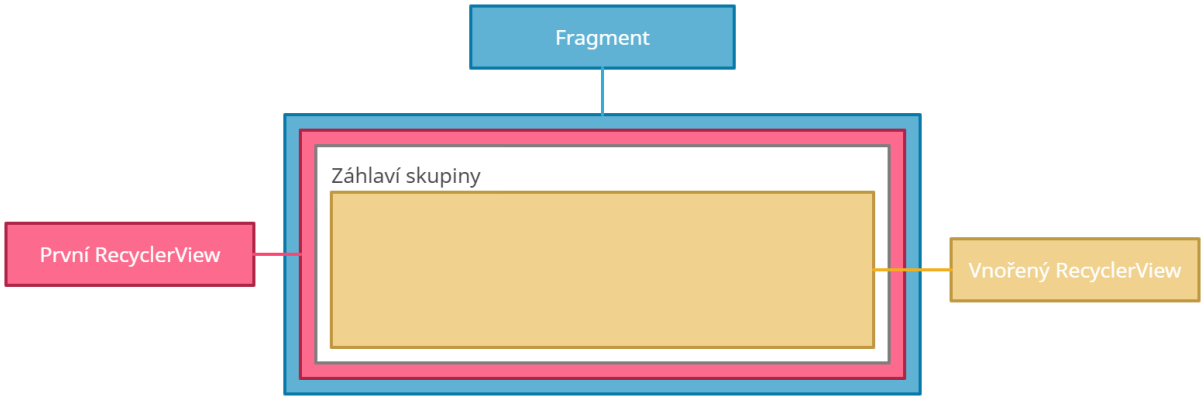
[2]Tato pravidla zahrnují například odsazení komponent, barvy aplikace, tvary, přechody atd.

zastřešuje komponenty `ImageView` pro ikony a komponenty `TextView` pro informace o začátku první pracovní aktivity a odpracovaných hodinách v daný den, které jsou vedle grafu. Naspodu fragmentu je další komponenta rozložení `CardView`, ve které jsou tlačítka aplikace. V průběhu praxe byl do tohoto fragmentu přidán ještě rámeček `CardView` s textovým polem `TextView`, který se zobrazuje při přerušení pracovní doby s časem, kdy byla práce přerušena. Více o implementaci funkcionalit tohoto fragmentu je uvedeno v podkapitole 8.3.4 a výsledný vzhled lze vidět na obrázku 8.3.

Fragment obrazovky „Přehled“ obsahuje komponentu `TabLayout` pro zobrazení karet dle návrhu uvedeného v kapitole 7.2 a viditelného na obrázku 7.1b. Pro správnou funkci této komponenty bylo nutné přidat i komponentu `ViewPager`, která poskytuje navigaci mezi fragmenty pomocí gest. Následně jsem pro karty zobrazované pomocí komponenty `TabLayout` vytvořil dva další fragmenty. Jeden pro denní přehled, který lze vidět na zmíněném obrázku a druhý pro týdenní a měsíční přehled. Pro správnou funkčnost komponenty `ViewPager` bylo nutné naprogramovat adaptér, neboli třídu dědící z `FragmentPagerAdapter`. Tento adaptér udržuje dané fragmenty ve správci fragmentů a díky němu se tyto části uživatelského rozhraní správně zobrazují podle vybrané karty v `TabLayout` přes komponentu `ViewPager`.

Na obrazovce denního přehledu byl navrhován seznam ukončených aktivit seskupených dle data (viz obrázek návrhu 7.1b). Pro tuto obrazovku jsem se rozhodl využít komponentu `RecyclerView`, která se dá využít jako nekonečný seznam a umožňuje definici, jak má položka v seznamu vypadat. Jelikož jsou položky (aktivity) seskupeny podle data, musel jsem použít tyto komponenty dvě. Na obrázku 8.2 můžeme vidět, jak jsou tyto komponenty do sebe umístěny. Následně jsem k oboum komponentám `RecyclerView` vytvořil XML soubory, které definují komponenty položek a jejich vzhled. Jeden soubor obsahuje `TextView` pro záhlaví (datum) skupiny aktivit, a vnořenou komponentu `RecyclerView` pro zobrazení položek, reprezentující aktivity seskupené dle data. Druhý soubor obsahuje komponenty `TextView` pro zobrazení dat a `ImageView` pro zobrazení ikony statusu schválení. Pro vykreslování dat v těchto komponentách bylo nutné naprogramovat dva adaptéry, tedy třídy dědící ze třídy `RecyclerView.Adapter` a ke každému adaptéru naprogramovat podtřídu dědící ze třídy `RecyclerView.ViewHolder`. Třída `RecyclerView.Adapter` poskytuje vazbu mezi daty a zobrazením a `RecyclerView.ViewHolder` vyhledává a udržuje odkazy na komponenty zobrazení. Ve vnořeném `RecyclerView.Adapter` je navíc nastaveno otevření dialogového fragmentu pro detail aktivity. Výsledek naprogramovaných komponent `RecyclerView` můžeme vidět na obrázku 8.3b.

Níže uvedený výpis 8.1 obsahuje ukázku nejdůležitějších metod třídy adaptéru první komponenty `RecyclerView`, která slouží pro zobrazení skupin aktivit. Pro inicializaci objektu třídy tohoto adaptéru jsem vytvořil konstuktor (viz řádky 1 – 6), který očekává data stažená skrze API a správce fragmentů `FragmentManager`. Správce fragmentů je parametrem konstruktoru adaptéru vnořené komponenty `RecyclerView`, jelikož je potřeba pro možnost otevření dialogového okna detailu aktivity kliknutím na danou aktivitu. Po vytvoření instance adaptéru je automaticky volána metoda `OnCreateViewHolder`, kterou můžeme vidět na řádcích 8 – 13. Na řádku 10 se v této metodě nejdříve



Obrázek 8.2: Schéma použití komponent `RecyclerView`

pomocí třídy `LayoutInflater` uloží instance definovaného rozložení v XML souboru do proměnné typu `View`. Tato instance rozložení je poté parametrem volaného konstruktoru na řádku 11. Voláním konstruktoru třídy `DailyOverviewViewHolder` je vytvořena instance této třídy, ve které jsou inicializovány komponenty v daném rozložení pomocí metod `FindViewById`. Následně je instance třídy `DailyOverviewViewHolder` vrácena pro další využití v následující automaticky volané metodě `OnBindViewHolder`, kterou můžeme vidět na řádcích 14 – 23. V této metodě se na řádku 18 do proměnné typu `LinearLayoutManager` uloží instance správce lineárního rozložení, jenž udává, jak se položky ve vnořené komponentě `RecyclerView` mají zobrazovat. V tomto případě je nastaveno, aby se položky zobrazovaly vertikálně za sebou. Následně je na řádku 19 tento správce rozložení nastaven vnořené komponentě `RecyclerView`. Na řádku 20 se nastaví zobrazovaný text v komponentě `TextView`, neboli záhlaví skupiny aktivit. Poté se do kolekce `List<ActivityViewModel>` přiřadí na řadku 21 aktivity ve formě viewmodelů, které spadají do dané skupiny. Na posledním řádku 22 se vytvoří instance adaptéru vnořené komponenty `RecyclerView` a následně je instance adaptéru nastavena této komponentě. Pomocí adaptéru vnořené komponenty `RecyclerView` jsou poté vykresleny aktivity spadající do dané skupiny.

```
1  public DailyOverviewAdapter(Dictionary<DateTime,List<ActivityViewModel>> data,
       FragmentManager fragmentManager)
2  {
3      this.data = data;
4      this.fragmentManager = fragmentManager;
5      this.dates = data.Keys.ToArray();
6  }
7
```

```
8  public override RecyclerView.ViewHolder OnCreateViewHolder(ViewGroup parent, int
       viewType)
9  {
10     View itemView = LayoutInflater.From(parent.Context).Inflate(Resource.Layout.
           daily_overview_row_group, parent, false);
11     return new DailyOverviewViewHolder(itemView);
12 }
13
14 public override void OnBindViewHolder(RecyclerView.ViewHolder viewHolder, int
       position)
15 {
16     var item = dates[position];
17     var holder = viewHolder as DailyOverviewViewHolder;
18     LinearLayoutManager linearLayoutManager = new LinearLayoutManager(holder.item.
           Context, LinearLayoutManager.Vertical, false);
19     holder.item.SetLayoutManager(linearLayoutManager);
20     holder.date.Text = item.ToString("dd.MM.yyyy");
21     List<ActivityViewModel> memberItems = data[item];
22     holder.item.SetAdapter(new ActivityAdapter(memberItems, fragmentManager));
23 }
```

Výpis 8.1: Ukázka metod třídy adaptéru komponenty `RecyclerView`

Fragment pro týdenní a měsíční přehled práce je až na jednu informaci o stravenkách totožný, proto jsem vytvořil pouze jeden a podle informace, zda je vytvářený fragment pro měsíční či týdenní přehled, zadané v konstruktoru při vytváření instance tohoto fragmentu, se zobrazuje nebo skrývá řádek s počtem stravenek. Tato informace ještě určuje, co se má zobrazovat v horní liště pro přepínání období a jak se metody pro přepínání období mají chovat. Mimo jiné je v tomto fragmentu implementován i graf z balíčku Microcharts (viz 5.10).

Fragment obrazovky „Žádosti“ obsahuje stejně jako fragment předešlé obrazovky „Přehled“ komponenty `TabLayout` a `ViewPager`. Navíc využívá pro přepínání karet stejný adaptér. Mimo to je ve fragmentu i tlačítko `FloatingActionButton` pro otevření formuláře nové žádosti. Toto tlačítko je naprogramováno, aby se zobrazovalo jen při aktivní kartě „Nadcházející“ (viz 7.1c) a při přechodu na kartu „Historie“ se skrývá. Pro karty byla vytvořena jedna třída fragmentu, který obsahuje komponentu `ListView`. Při vytváření instance třídy fragmentu se v konstruktoru udává informace typu `Enum`[3] pro volání správné služby API. Pro definici a vzhled položek v seznamu komponenty `ListView` byl sepsán soubor XML. Následně bylo nutné naprogramovat adaptér pro tuto komponentu, neboli

[3]Informace je future pro nadcházející žádosti nebo past pro staré žádosti.

třídu dědící ze třídy `BaseAdapter`, která dosadí data do komponent definovaných v XML souboru a vytvoří z nich seznam, který se následně zobrazí uživateli.

Posledním hlavním fragmentem v menu je fragment obrazovky „Profil“. V tomto fragmentu je implementován graf z balíčku Microcharts, komponenty `TextView` pro zobrazení základních dat přihlášeného uživatele a komponenta `ListView` sloužící jako menu s volbami pro změnu hesla, otevření obrazovky se seznamem lokací a odhlášení.

Obrazovky zobrazující formuláře pro novou žádost, změnu hesla, novou lokaci a obrazovky se seznamem lokací a detailem aktivity jsem implementoval jako dialogová okna, naprogramoval jsem tedy třídy dědící ze třídy `DialogFragment`. Tato dialogová okna jsou nastavena, aby se zobrazila při kliknutí na příslušná tlačítka a v případě detailu aktivity je tento dialog zobrazen po kliknutí na položku v seznamu komponenty `RecyclerView` popsané výše. Jelikož se dialogová okna v systému Android zobrazují jako menší okénka, přenastavil jsem ve třídách těchto dialogů šířku a výšku, aby se zobrazily přes celou obrazovku zařízení a také jsem těmto obrazovkám nastavil animace. Dialogový fragment se seznamem lokací je řešen obdobně jako fragment se seznamem žádostí.

8.3.2 Využití návrhového vzoru Model-View-ViewModel

Protože jsou zobrazované informace založeny na datech databáze a bylo počítáno s tím, že se tato data dostanou do aplikace pomocí API, byl následující úkol připravit komponenty rozložení na data, která budou přicházet ve formě objektů datových modelů. V průběhu tohoto úkolu se konala konzultace a bylo nám navrženo, ať se pro jednodušší práci inspirujeme návrhovým vzorem model-view-viewmodel. Tento vzor při využití v plném rozsahu odděluje data, stav aplikace a uživatelské rozhraní. Museli jsme tedy s kolegou vytvářejícím API zanalyzovat navržené obrazovky a zhodnotit jaké viewmodely jsou pro aplikaci relevantní. Tyto viewmodely byly potom kolegou vytvořeny ve sdílené knihovně tříd pro možnost využití v rámci celého projektu. Ačkoliv tyto viewmodely využívám v kódu aplikace, nejsou využity v plném rozsahu podle tohoto návrhového vzoru, jelikož nemění uživatelské rozhraní aplikace a jsou využívány pouze pro zasílání a stahování dat skrze API a k jejich držení. Plné využití tohoto návrhového vzoru je plánováno do budoucna.

8.3.3 Propojení s rozhraním API

Následující částí vývoje bylo propojení aplikace s API. Pro kontrolu funkčnosti bylo úkolem předvést komunikaci aplikace s API při přihlášení do aplikace. Na konzultaci jsme se dohodli, že budeme využívat zabezpečenou komunikaci pomocí tokenu. Token je zašifrovaný řetězec znaků s časovou platností, vygenerovaný službou API při přihlášení uživatele a udržující informace o uživateli (v případě tohoto systému e-mail a identifikátor uživatele) a je za potřebí k dalšímu volání služeb API pro autentizaci uživatele. Pro tuto problematiku jsem si musel nastudovat přístup do zabezpečeného úložiště systému Android pro ukládání tohoto tokenu. Pro komunikaci s API jsem musel nejdříve vytvořit třídu, která bude obsluhovat připojení k API serveru a metody pro zasílání a přijímání

dat. Ve sdílené vrstvě projektu jsem tedy vytvořil s pomocí zkušenějšího kolegy z firmy následující třídy:

- **`ApiClient`** - Třída obsluhující připojení k serveru API a obsahující obecné statické asynchronní metody pro POST a GET, v rámci těchto metod je prováděna i serializace a deserializace objektů.
- **`ApiClientService`** - Třída obsahující asynchronní statické metody pro konkrétní využití v aplikaci, z těchto metod jsou volány obecné metody ze třídy `ApiClient` s požadovaným URI služby API.

V následujícím výpisu 8.2 můžeme vidět část třídy `ApiClient`. Na řádcích 3 – 8 je definován statický konstruktor třídy. Tento konstruktor je volán automaticky před voláním metod této třídy a pomocí něj je inicializován HTTP klient, který slouží pro zasílání požadavků a přijímání odpovědí pomocí protokolu HTTP, vytvořením instance třídy `HttpClient`. Dále je tomuto klientu přiřazena základní adresa API serveru a nastaven formát serializace přijímaných dat na JSON (viz řádky 6 a 7). Tato instance je poté po celou dobu používání aplikace udržována ve statickém atributu `client`. Na řádcích 10 – 15 můžeme vidět ukázku obecné asynchronní metody pro POST. Metoda slouží pro zasílání obecných dat typu `Q` a přijímání jiných obecných dat typu `T`. Tato metoda má deklarované parametry `requestUri` typu `string` pro přesnou adresu služby API, `isAuthorized` typu `bool` pro informaci, zda jde o autorizovaný přístup ke službě, `token` typu `string` pro autentizační token a obecný model `Q` pro zasílaná data. V této metodě se jako první vyhodnotí, zda se jedná o autorizovaný přístup, což můžeme vidět na řádku 12 a podle toho se do hlavičky HTTP klienta přidá token. Na řádku 13 je poté zavolána asynchronní metoda klienta `PostAsJsonAsync`. Tato metoda serializuje data ve formě objektu do formátu JSON a pomocí HTTP metody POST je poté odešle na adresu služby API, která je parametrem metody, výsledek této metody je poté uložen do proměnné typu `HttpResponseMessage`. Na řádku 14 je tento výsledek parametrem volané metody `HandleResponse`, kterou můžeme vidět na řádcích 17 – 25. Tato metoda slouží pro deserializaci přijatých dat a uložení odpovědi do objektu generické třídy `ServiceResult<T>`. Tato třída má atributy pro informaci o úspěšnosti požadavku, výsledek požadavku, neboli přijímaná data a zprávu odezvy HTTP požadavku (viz řádky 22 a 24).

```
1  private static readonly HttpClient client;
2
3  static ApiClient()
4  {
5      client = new HttpClient();
6      client.BaseAddress = new Uri(@"https://dochazkovysystemwebapi.azurewebsites.
           net/");
```

```
7       client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("
            application/json"));
8   }
9
10  public static async Task<ServiceResult<T>> PostDataAsync<T, Q>(string requestUri,
        bool isAuthorized, string token, Q model)
11  {
12      client.DefaultRequestHeaders.Authorization = isAuthorized ? new
            AuthenticationHeaderValue("Bearer", token) : null;
13      HttpResponseMessage response = await client.PostAsJsonAsync(requestUri, model)
            ;
14      return await HandleResponse<T>(response);
15  }
16
17  private static async Task<ServiceResult<T>> HandleResponse<T>(HttpResponseMessage
        response)
18  {
19      if(response.IsSuccessStatusCode)
20      {
21          var responseContent = await response.Content.ReadAsAsync<T>();
22          return new ServiceResult<T> { Successful = true, Result = responseContent,
                Message = response.ReasonPhrase };
23      }
24      return new ServiceResult<T> { Successful = false, Result = default, Message =
            response.ReasonPhrase };
25  }
```

Výpis 8.2: Ukázka metod třídy `ApiClient`

V projektu aplikace byla vytvořena ještě třída `ApiConnector` pro volání metod třídy `ApiClientService`. Mimo to metody této třídy přistupují do zabezpečeného úložiště, kde se po přihlášení ukládá autentizační token a v případě jiných metod je zasílán pro autentizaci uživatele. Metody ze třídy `ApiConnector` jsou dále volány ve třídách obrazovek aplikace pro zasílání a přijímání dat. Obrazovka přihlášení využívá pro spojení s API třídu `LoginTask` dědící ze třídy `AsyncTask`, která spouští asynchronně kód na pozadí. U ostatních obrazovek jsem využíval asynchronní metody.

8.3.4 Implementace funkcionalit obrazovky „Docházka“

Jelikož obrazovka „Docházka“ (viz podkapitola 7.1) skrývá hlavní funkcionalitu aplikace, strávil jsem nejvíce času při její implementaci. Do této obrazovky jsou po přihlášení skrze API stažena data, která jsou důležitá pro zobrazení uživateli ale i pro správnou funkčnost zaznamenávání docházky.

8.3.4.1 Kontrola polohy

Již dříve zmíněnou žádanou funkcionalitou byla kontrola polohy zaměstnanců, tato kontrola by měla probíhat ve chvíli kdy chce zaměstnanec začít pracovat nebo práci ukončit, či přerušit. Pro kontrolu lokace jsou důležitá data schválených lokací zaměstnance nebo lokace pobočky získána skrze API. Ve výpisu 8.3 můžeme vidět že kontrola lokace probíhá v asynchronní metodě, ve které se pomocí Xamarin.Essentials[4] na řádcích 5 – 9 získá lokace zařízení. Pokud tato lokace vrácená metodou `GetLocationAsync` není při větvení na řádku 10 z nějakého důvodu prázdná (například kvůli špatnému signálu), projde se následně v cyklu seznam lokací uložený v objektu viewmodelu hlavní obrazovky a pomocí metody na řádku 14 pro výpočet vzdálenosti se poté přibližně vyhodnotí, jestli je zaměstnanec na určeném místě podle pracovního plánu. Tato vzdálenost byla prozatím nastavena na 100 metrů. Metoda následně vrací identifikátor lokace pro vkládání nové aktivity skrze API. V případě, že uživatel nepovolí aplikaci využívat lokaci zařízení, aplikace mu nedovolí začít pracovat.

```
1  private async Task<string> CheckLocation()
2  {
3      try
4      {
5          var myLoc = await Geolocation.GetLocationAsync(new GeolocationRequest
6          {
7              DesiredAccuracy = GeolocationAccuracy.Medium,
8              Timeout = TimeSpan.FromSeconds(30)
9          });
10         if (myLoc != null)
11         {
12             foreach (LocationViewModel reqLoc in homeScreenData.WorkLocations)
13             {
14                 if (myLoc.CalculateDistance(Decimal.ToDouble(reqLoc.Lat), Decimal.
                       ToDouble(reqLoc.Long), DistanceUnits.Kilometers) < 0.1)
15                 {
16                     return reqLoc.Id;
```

[4]Xamarin.Essentials poskytuje základní multiplatformní služby pro mobilní aplikace. V případě této aplikace byla využita služba pro přístup do zabezpečeného úložiště a geolokační služby.

```
17                 }
18             }
19         }
20     }
21     catch(Exception ex)
22     {
23         return null;
24     }
25
26     return null;
27 }
```

Výpis 8.3: Metoda pro kontrolu polohy

8.3.4.2 Aktualizace dat v reálném čase

Další metoda, která byla v této obrazovce implementována, byla pro aktualizaci zobrazovaných dat v reálném čase v případě běžící pracovní aktivity. Pro tyto účely jsem využil časovače, který je dostupný v platformě .NET. Tento časovač je nastaven, aby se metoda spouštěla každou sekundu. V této metodě se poté pomocí vláken aktualizují na pozadí data zobrazovaná grafem a textová pole reprezentující celkový odpracovaný čas. Pokud zaměstnanec přeruší pracovní dobu vybráním aktivity typu absence v dialogovém oknu pro přerušení pracovní doby, který se zobrazí kliknutím na tlačítko „přerušení pracovní doby“ (viz obrázek 8.3a), tento časovač se zastaví.

Na řádku 3 v uvedeném výpisu 8.4 se do atributu `timeWorked` přiřadí buď doba, která uběhla od začátku poslední aktivity, nebo hodnota součtu této doby s celkovým časem ukončených produktivních a neproduktivních aktivit v daný den. Záleží na tom, jestli zrovna běží první aktivita v daný den a nebo například aktivita po obědové pauze. Na řádku 6 je poté ve vlákně aplikace, které se stará o změny uživatelského rozhraní, změněn text v komponentě `TextView` reprezentující odpracovaný čas. Na řádku 7 je následně volána metoda, která se stará o vykreslování grafu. V této metodě se poté využívá hodnota `timeWorked` pro správné vykreslení grafu.

```
1 private void WorkTimerElapsed(object sender, System.Timers.ElapsedEventArgs e)
2 {
3     timeWorked = homeScreenData.ProductiveTime.HasValue ? homeScreenData.
          ProductiveTime.Value + (DateTime.UtcNow - homeScreenData.
          StartOfLastActivity.Value) : DateTime.UtcNow - homeScreenData.
          StartOfLastActivity.Value;
4
```

```
5       Activity.RunOnUiThread(() => {
6           tvOverallTime.Text = timeWorked.ToString(@"hh\:mm");
7           DrawDonutChart();
8       });
9   }
```

Výpis 8.4: Metoda časovače

8.3.4.3 Nastavení tlačítek

V rámci obrazovky „Docházka“ bylo pro správnou funkčnost potřebné správně blokovat tlačítka a také nastavit vlastnosti hlavního tlačítka (na obrázku 8.3a tlačítko s textem „pokračovat“), kterým se zahajují a ukončují aktivity. Pro toto nastavení jsou nezbytná získaná data přes API. Mezi tato data patří čas začátku první pracovní aktivity v daný den, čas začátku poslední neukončené aktivity, identifikátor aktivity, typ aktivity a informace o tom, jestli si může vzít uživatel půldenní volno.

Ve výpisu 8.5 na řádcích 1 a 2 můžeme vidět, že tlačítka, přes které si může zaměstnanec vzít celodenní nebo půldenní dovolenou jsou blokována jednoduše prostřednictvím k tomu určené informace, která je součástí dat stažených z API serveru. Tato informace je určena na základě existence plánu dovolené na daný den. U blokace tlačítka celodenního volna se z logických důvodů vyhodnocuje navíc informace, jestli zaměstnanec již začal pracovní aktivitu v daný den. Tlačítko pro přerušení práce je zakázáno pouze před začátkem a po ukončení pracovní aktivity v daný den (viz řádek 3).

Tlačítko pro začátek práce je komplikovanější, neboť se mění jeho vlastnosti v průběhu používání aplikace. Pokud nebyla zahájena žádná aktivita v daný den, tlačítko má nastavený text jako „začátek práce“ a jeho barva je zelená toto můžeme vidět na řádcích 5 – 10 ve zmíněném výpisu, v případě kliknutí na toto tlačítko je zkontrolována poloha zaměstnance a zaslána spolu s tokenem a časem na server API, následně jsou staženy nová data a spuštěn časovač pro aktualizaci dat (viz podkapitola 8.3.4.2). Po začátku aktivity se změní barva tlačítka na červenou a text tlačítka na „konec práce“ (viz řádky 22 – 28). Pokud uživatel práci přeruší tlačítkem „přerušení pracovní doby“ (viz obrázek 8.3a), změní se barva hlavního tlačítka na oranžovou a text na „pokračovat v práci“ (viz řádky 11 – 21). Toto tlačítko má poté funkci ukončení aktivity přerušení a vytvoření nové pracovní aktivity. Ukončením pracovní aktivity kliknutím na červené tlačítko „konec práce“ se z důvodu vymyšlené logiky zablokují všechna tlačítka aplikace.

```
1 btnHalfDay.Enabled = homeScreenData.CanTakeADayOff;
2 btnFullDay.Enabled = homeScreenData.CanTakeADayOff && !homeScreenData.StartOfDay.
      HasValue;
3 btnCustom.Enabled = homeScreenData.StartOfLastActivity.HasValue;
```

```
4  btnStart.Enabled = !homeScreenData.StartOfDay.HasValue || homeScreenData.
       ActivityId != null;
5  if (homeScreenData.StartOfDay.HasValue && homeScreenData.ActivityId == null && !
       homeScreenData.StartOfLastActivity.HasValue)
6  {
7      btnStart.Text = "START";
8      btnStart.BackgroundTintList = Context.GetColorStateList(Resource.Color.
           button_start_background_color);
9      cvTimeOffNotification.Visibility = ViewStates.Gone;
10 }
11 if (homeScreenData.StartOfLastActivity.HasValue && homeScreenData.ActivityId !=
       null && homeScreenData.TypeOfActivity != ActionType.Productive)
12 {
13     btnStart.Text = "POKRAČOVAT V PRÁCI";
14     btnStart.BackgroundTintList = Context.GetColorStateList(Resource.Color.
           button_continue_background_color);
15     tvTimeOffSince.Text = homeScreenData.StartOfLastActivity.Value.ToString("t");
16     cvTimeOffNotification.Visibility = ViewStates.Visible;
17     if (homeScreenData.TypeOfActivity == ActionType.Absence)
18         workTimer.Stop();
19     else
20         workTimer.Start();
21 }
22 if (homeScreenData.StartOfLastActivity.HasValue && homeScreenData.ActivityId !=
       null && homeScreenData.TypeOfActivity == ActionType.Productive)
23 {
24     btnStart.Text = "KONEC";
25     btnStart.BackgroundTintList = Context.GetColorStateList(Resource.Color.
           button_end_background_color);
26     workTimer.Start();
27     cvTimeOffNotification.Visibility = ViewStates.Gone;
28 }
```

Výpis 8.5: Ukázka nastavení tlačítek při načtení dat skrze API

8.3.5 Splash screen

Splash screen neboli uvítací obrazovka byla naprogramována především pro kontrolu přihlášení při zapnutí aplikace. Při zobrazení této obrazovky je volána metoda pro kontrolu tokenu pomocí služby

API. Pokud token v zabezpečeném úložišti existuje a je platný, je službou API obnoven a přepíše stávající token, následně je zobrazena hlavní obrazovka aplikace, pokud ne, zobrazí se obrazovka pro přihlášení do systému. Implementací této obrazovky se tak eliminovala nutnost přihlášení při každém spuštění aplikace.

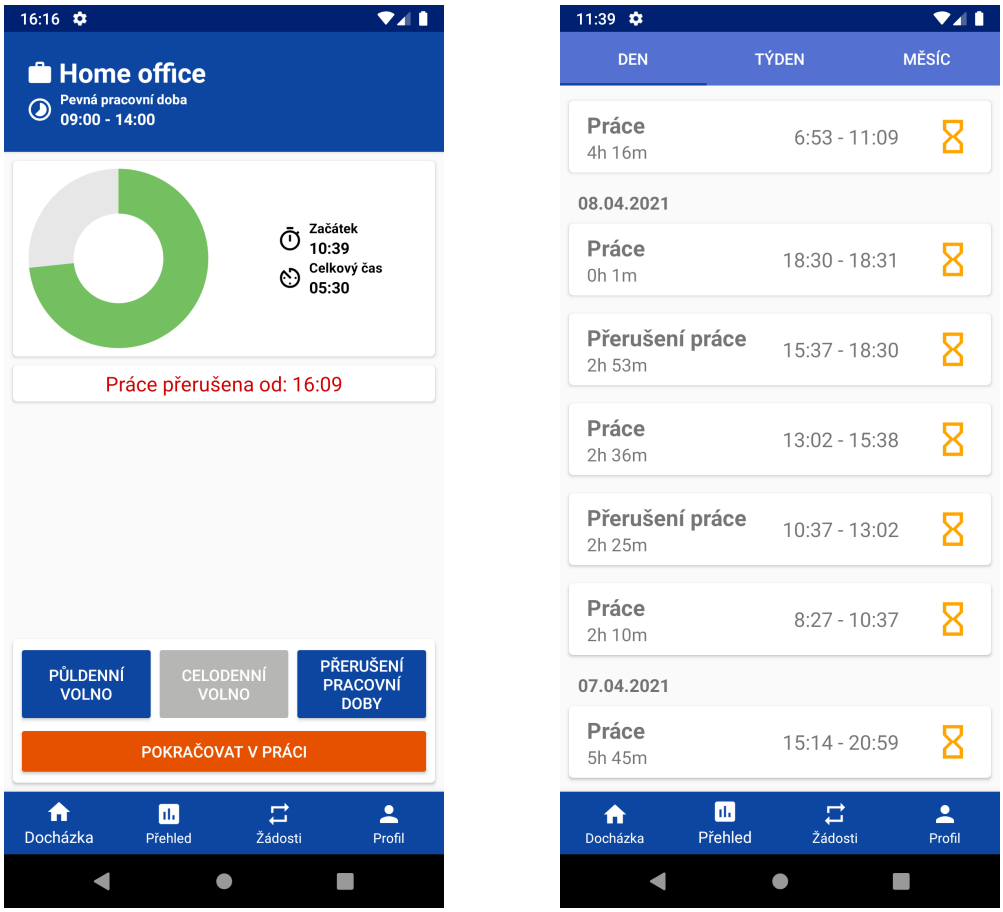
8.3.6 Poslední fáze vývoje

V posledních fázích vývoje aplikace jsem se zaměřoval hlavně na úpravy designu a refaktorování[5]. Jedna z těchto fází byla i aktualizace úrovně API, která zahrnovala aktualizaci zastaralých komponent knihovny Android Support na komponenty knihovny AndroidX. Pro vylepšení designu aplikace jsem se v pozdějších fázích rozhodl využít knihovnu Shimmer vytvořenou společností Facebook pro Android aplikace [25]. Tato knihovna poskytuje efekt třpytění, který jsem využil jako indikátor načítání dat v seznamech žádostí, lokací a denním přehledu aktivit.

8.3.7 Finální podoba aplikace

Na následujících obrázcích 8.3 můžeme vidět dvě obrazovky z finální aplikace. Na obrázku 8.3a je ukázka hlavní obrazovky „Docházka“. Tato obrazovka se zobrazí při přihlášení do aplikace. Tlačítka této obrazovky slouží pro zaznamenávání docházky a prstencový graf se plní na základě uběhlého času produktivních aktivit (viz podkapitola 8.3.4.2). Na obrázku je zrovna zachycen stav aplikace, kdy zaměstnanec práci přerušil pomocí tlačítka pro přerušení pracovní doby a automatická aktualizace dat na obrazovce je zastavena. Pokud by zaměstnanec pracoval více než by měl odpracovat za den, graf se bude plnit červenou barvou. Na druhém obrázku 8.3b je možné vidět vykreslování odpracovaných aktivit pomocí naprogramovaných komponent `RecyclerView` popsaných výše (viz podkapitola 8.3.1).

[5]Refaktorování je disciplinovaný proces provádění změn v softwarovém systému takovým způsobem, že nemají vliv na vnější chování kódu, ale vylepšují jeho vnitřní strukturu s minimálním rizikem vnášení chyb. [24]



(a) Obrazovka „Docházka“ (b) `RecyclerView`

Obrázek 8.3: Ukázka finální aplikace

Kapitola 9

Nasazení a testování mobilní aplikace

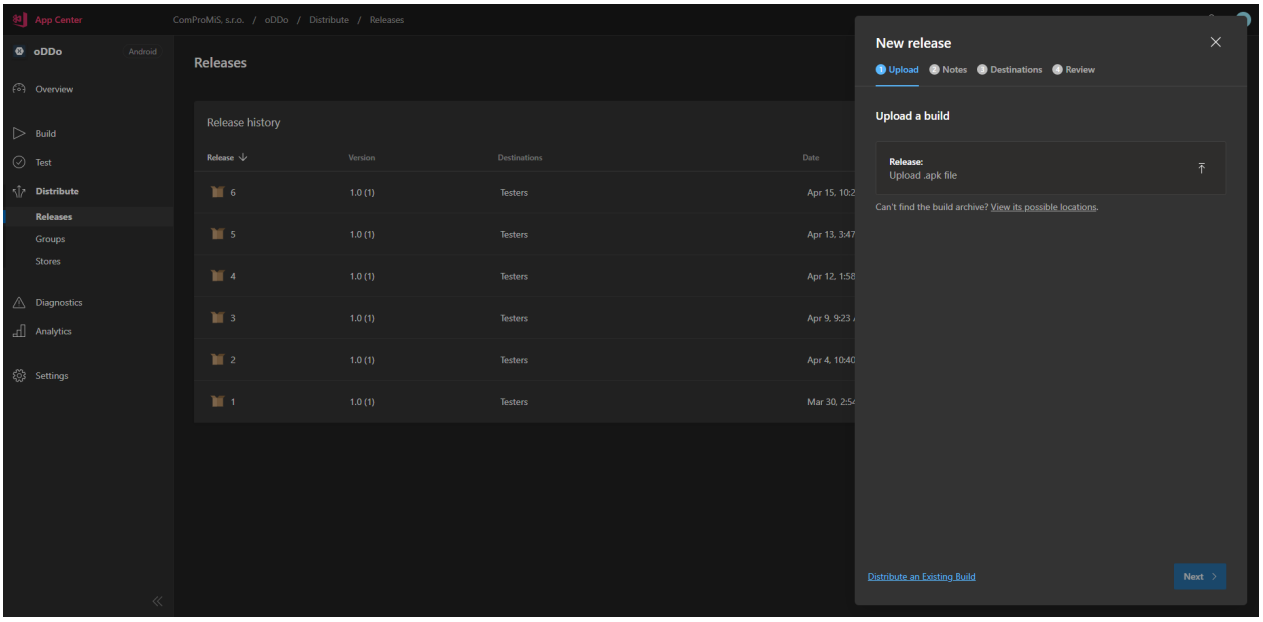
Jakmile byla dokončena funkční verze aplikace, následovalo nasazení do testovacího provozu v rámci firmy. Nejprve musely být vyřešeny chyby, které nastaly při archivaci aplikace v nástroji Visual Studio. Po archivaci musela být aplikace podepsána, aby bylo možné ji dále distribuovat. Podpis aplikace znamená, že jsem si musel vytvořit certifikát pro identifikaci mě jako autora aplikace a k tomuto certifikátu vytvořit klíč, pomocí kterého je poté certifikát přidán do instalačního souboru aplikace. Tento certifikát je důležitý, neboť systém Android jej vyžaduje při instalaci jakékoliv aplikace, bez tohoto certifikátu by nešla aplikace nainstalovat. Aplikace docházkového systému byla distribuována skrze službu Visual Studio App Center (viz obrázek 9.1), do které jsem získal přístup v rámci firmy. V této službě jsem vytvořil testovací skupinu složenou z kolegů firmy a vývojového týmu disponujícími mobilním zařízením se systémem Android. Při vydávání aplikace je nahraný instalační soubor aplikace získán archivací v nástroji Visual Studio a je vybrána testovací skupina. Lidem z této skupiny je zaslán službou e-mail, který obsahuje odkaz na stažení dané verze aplikace.

Osobně zařízení s Androidem nevlastním a aplikaci jsem testoval pouze v emulátoru, který je dostupný v nástroji Visual Studio, byla pro mě tedy důležitá zpětná vazba od kolegů. V průběhu testování se vyskytly různé chyby, které nebyly viditelné při vývoji aplikace. Chyby se týkaly zejména špatného zobrazování uživatelského rozhraní aplikace na různých zařízeních. Vyskytly se ovšem i chyby v nestabilitě aplikace a ve špatném zobrazování dat. V rámci testování bylo vydáno šest testovacích verzí aplikace, kdy byly tyto chyby řešeny a došlo k aktualizaci části aplikace na základě zpětné vazby a požadavků změn.

9.1 Výsledný stav aplikace

Mnou vyvíjená aplikace je ve výsledném stavu funkční, ale není zcela dokončena. Jelikož z časových důvodu nebyla možnost aplikaci testovat delší dobu a více lidmi, je možné, že nebyly nalezeny všechny chyby. Do budoucna je potřeba se zaměřit na úpravu kódu aplikace. Jedním z nedořešených problémů je i zamezení zneužívání systému, například navázáním jednoho mobilního zařízení

k jednomu účtu. V budoucnu je možné, že se systém bude rozšiřovat i pro zadávání práce, nebo pro reporty pracovních cest.



Obrázek 9.1: Služba App Center

Kapitola 10

Závěr

V počátcích praxe mi při návrhu nového informačního systému pomohly znalosti z předmětů Úvod do softwarového inženýrství, Úvod do databázových sytémů a Vývoj informačních systémů. Nejvíce jsem při vývoji využil znalosti z předmětu Tvorba aplikací pro mobilní zařízení 2. V tomto předmětu jsem se naučil nejen jak postupovat při vývoji aplikace pro mobilní systém Android, ale také pro mě nebylo novinkou verzování v systému Git.

Během praxe mi bylo rozšířeno povědomí o cloudových službách. V tomto odvětví mi pomohla zejména práce s platformou Azure, se kterou jsem se setkal poprvé a která byla využívána po celou dobu praxe. Dále jsem nabyl nových zkušenosti s verzováním projektu pomocí úložiště Git prostřednictvím nástroje Azure Repos. Jako další novou zkušeností bylo nasazení aplikace do reálného testovacího provozu pomocí služby App Center. Nových znalostí jsem nabyl i při programování aplikace, obzvlášť při práci s platformou Xamarin, se kterým jsem se taktéž setkal prvně. Rozšířeny byly i mé zkušenosti s programovacím jazykem C#. Pro mě nejzásadnější novinkou byla práce s Entity Framework. Obzvlášť vytvoření databáze s pomocí přístupu Code First.

Odbornou praxi ve firmě ComProMiS s.r.o. vnímám jako cennou zkušenost. Největší zkušeností pro mě byla práce v týmu na reálném projektu. Během praxe jsem se dozvěděl o nových technologiích a postupech používaných při vývoji mobilní aplikace pro systém Android. Při analýze, návrhu a vývoji jsem získal i nové vědomosti o tom, jak taková firma funguje.

Literatura

1. *Compromis, O nás* [online] [cit. 2021-04-19]. Dostupné z: https://www.compromis.cz/new/index.
2. *Zákoník práce (Česko, 2006)* [online] [cit. 2021-04-19]. Dostupné z: https://cs.wikipedia.org/wiki/Z%C3%A1kon%C3%ADk_pr%C3%A1ce_(%C4%8Cesko,_2006).
3. *Microsoft Azure* [online] [cit. 2021-04-19]. Dostupné z: https://cs.wikipedia.org/wiki/Microsoft_Azure.
4. *Middleware* [online] [cit. 2021-04-19]. Dostupné z: https://cs.wikipedia.org/wiki/Middleware.
5. *Co je SaaS?* [Online] [cit. 2021-04-19]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-saas/.
6. *Co je PaaS?* [Online] [cit. 2021-04-19]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-paas/.
7. *Co je IaaS?* [Online] [cit. 2021-04-19]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-iaas/.
8. *Visual Studio - centrum aplikací* [online] [cit. 2021-04-19]. Dostupné z: https://visualstudio.microsoft.com/cs/app-center/.
9. *Microsoft Visual Studio* [online] [cit. 2021-04-19]. Dostupné z: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio.
10. *Microsoft SQL Server Management Studio* [online] [cit. 2021-04-19]. Dostupné z: https://en.wikipedia.org/wiki/SQL_Server_Management_Studio.
11. *MS SQL Server v kostce* [online] [cit. 2021-04-19]. Dostupné z: https://www.mssql.cz/post/ms-sql-server-v-kostce.
12. *What is Entity Framework?* [Online] [cit. 2021-04-19]. Dostupné z: https://www.entityframeworktutorial.net/what-is-entityframework.aspx.
13. *C Sharp* [online] [cit. 2021-04-19]. Dostupné z: https://cs.wikipedia.org/wiki/C_Sharp.

14. *Co je Xamarin?* [Online] [cit. 2021-04-19]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/get-started/what-is-xamarin.

15. *Proto.io* [online] [cit. 2021-04-19]. Dostupné z: https://en.wikipedia.org/wiki/Proto.io.

16. *Creately* [online] [cit. 2021-04-19]. Dostupné z: https://en.wikipedia.org/wiki/Creately.

17. *Microcharts* [online] [cit. 2021-04-19]. Dostupné z: https://github.com/dotnet-ad/Microcharts.

18. *NuGet* [online] [cit. 2021-04-19]. Dostupné z: https://www.nuget.org/.

19. *Iconify* [online] [cit. 2021-04-19]. Dostupné z: https://github.com/PragmaticIT/xiconify.

20. *Introduction to ASP.Net Identity 2.0* [online] [cit. 2021-04-19]. Dostupné z: https://www.c-sharpcorner.com/UploadFile/16101a/introduction-to-Asp-Net-identity-2-0/.

21. *Code first initializers a migrace - kompletní přehled* [online] [cit. 2021-04-19]. Dostupné z: https://www.dotnetportal.cz/clanek/8475/Code-First-initializers-a-migrace-kompletni-prehled.

22. *Material Design* [online] [cit. 2021-04-19]. Dostupné z: https://material.io/.

23. *Fragments* [online] [cit. 2021-04-19]. Dostupné z: https://developer.android.com/guide/fragments.

24. *Refaktorování* [online] [cit. 2021-04-19]. Dostupné z: https://cs.wikipedia.org/wiki/Refaktorov%C3%A1n%C3%AD.

25. *Shimmer* [online] [cit. 2021-04-19]. Dostupné z: https://facebook.github.io/shimmer-android/.